Applications of Genetic Algorithms and Quantum Genetic Algorithms on theoretical fully-autonomous road systems

University of Birmingham

Sam Barrett
sjb786@student.bham.ac.uk

November 25, 2020

Contents

Abstract

Chapter 1

Introduction

Chapter 2

Background

2.1 Genetic Algorithms

2.1.1 History

Genetic algorithms are optimisation techniques that employ the same rationale as classical Evolution seen in nature.

Genetic Algorithms can trace their origins back to the late 1960s when they were first proposed by John Holland, though he then referred to them as *Genetic Plans*[1]. Holland went on to write the first book on the subject titled *Adaptation in Natural and Artificial Systems*[1] in 1975. The field did not find much reception with Holland stating in the preface to the 1992 rerun:

> *"When this book was originally published I was very optimistic, envisioning extensive reviews and a kind of 'best seller' in the realm of monographs. Alas! That did not happen."*

However, in the early nineties, Genetic algorithms surged in popularity along with the area of Artificially Intelligent planning as a whole leading to Holland republishing his book and solidifying his position as the field's founder.

2.1.2 Definition

In a general sense, optimisation techniques work to find the set of parameters $\mathcal{P}$ that minimise an objective function $\mathcal{F}$. Genetic algorithms approach this by representing these sets as individuals in a population, P. Over the course of multiple generations, the best solutions are determined and promoted until termination criteria are met or the maximum number of generations is reached.

[1] This distinction was made to emphasise the *"centrality of computation in defining and implementing the plans"*[1]

As our candidates are essentially a collection of parameters to the function we are trying to optimise, we can extend our metaphor further by mapping each element of a individual to a *gene* in a individual's genome.

The representation we use in a GA is problem specific. Often we have to provide functions to facilitate the mapping between the problem specific set of possible solutions and the encoded genotype space in which we optimise. The most basic representation being a string of binary numbers.

An individual's characteristics and genetic information is normally encapsulated within the Phenotype. Here not only the genetic information of the Genotype but also additional information, such as fitness, is stored in order to prevent it from being re-calculated as often.

Genetic algorithms are both *probabilistically optimal* and *probabilistically complete*[2] meaning that: given infinite time, not only will the algorithm find *a* solution, (if one exists), it will find **the** optimal solution from the set of all possible solutions, $\mathcal{P}^*$.

Algorithm 1: Modern Generic Genetic Algorithm

Result: Best Solution, $p_{\texttt{best}}$
Generate initial population, P_0 of size n;
Evaluate fitness of each individual in P_0, $\{F(p_{0,1}, \ldots, p_{0,n})\}$;
while *termination criteria are not met* **do**
  Selection: Select individuals from P_t based on their fitness;
  Variation: Apply variation operators to parents from P_t to produce offspring;
  Evaluation: Evaluate the fitness of the newly bred individuals;
  Reproduction: Generate a new population P_{t+1} using individuals from P_t as well as the newly bred candidates.;
  t++
end
return $p_{\texttt{best}}$

As you can see from Figure 2.1 and Algorithm 1 the overall shape of GAs has not changed substantially over the course of the past 50 years. Being comprised of a series of operations that a starting population is piped through until criteria are met.

2.1.3 Genetic Operators

In the following section I will outline the various genetic operations that take place in a GA, they can be seen in Algorithm 1. Any operators used and analysed in Chapter 4 will have more detailed explanations of their process.

Selection

The selection procedure is the process by which the next generation of individuals is created from the current population. Individuals are selected

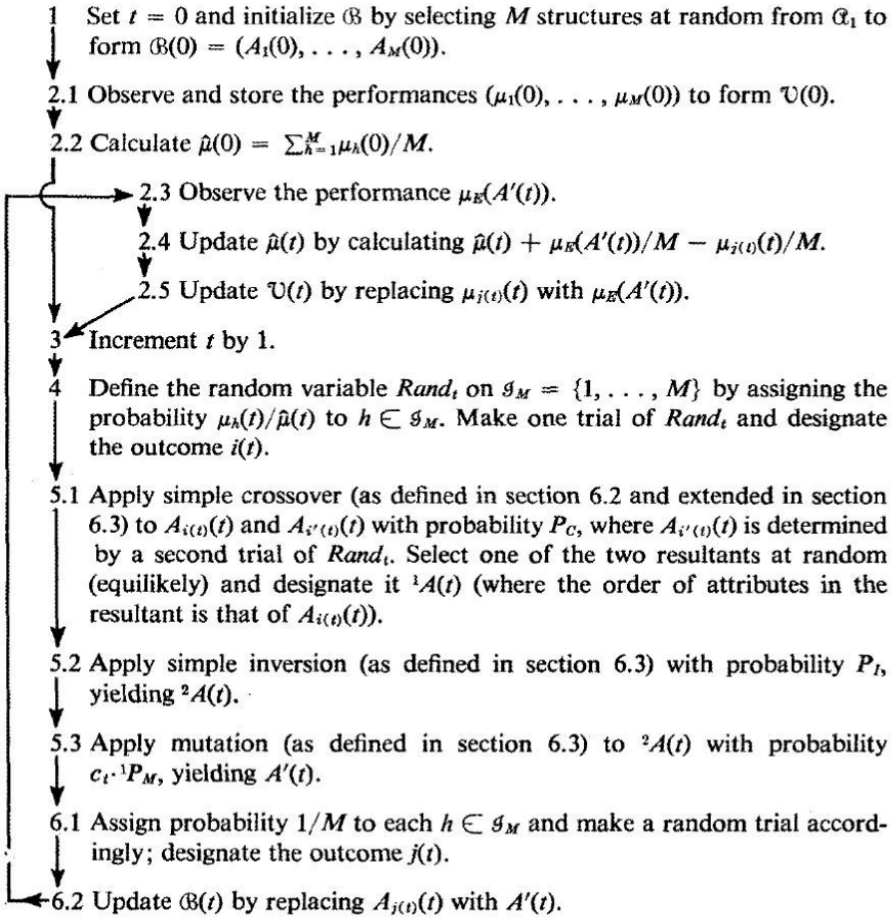

Figure 2.1: GA algorithm outlined in Holland's Original Book[1]

relative to their fitness as determined by the Objective (fitness) function $\mathcal{F}$.

Some methods select only the best solutions by fitness. Others employ a more stochastic approach, such as roulette wheel selection, to increase diversity and reduce complexity.

Fitness Proportional Selection Fitness proportional or Roulette wheel selection is a popular selection operator. It uses fitness to assign selection probability to each individual in a population.

The probability of selection for an individual i with fitness $\mathcal{F}(i)$ can be expressed mathematically as:

$$p_i = \frac{\mathcal{F}(i)}{\sum_{j=1}^{N} \mathcal{F}(j)} \tag{2.1}$$

Where N is the size of the population P

This is a simple approach but performs well and has very little performance overhead.

Variation

Variation in a GA is the process of altering the genome individuals to further explore the search space via stochastic local search. We perform this using two distinct sub-operations: Mutation and Crossover. Here we can view crossover as the *breeding* process and mutation as resembling the natural tendency for DNA to mutate over the course of generations.

Mutation In mutation we alter each gene with a set probability p_m known as the *mutation rate*. A standard value for a mutation rate is $\frac{1}{L}$ but it can fall anywhere in the range $p_m \in [\frac{1}{L}, \frac{1}{2}]$ where L is the length of the genome.

A low value for p_m a new individual which can be shown to be *close* to it's parents in the search space relative to their *Hamming distance*[2] if using a Binary coded GA. In real-coded GAs they can be shown to be close by the Euclidean distance between them.

In binary coded GAs we alter a given gene by *flipping* it's value. In real-coded GAs mutation operators include:

- Uniform Mutation
- Non-Uniform Mutation
- Gaussian Mutation

[2]Hamming Distance: The metric for comparing two binary data strings. The Hamming distance between two strings is the number of bit position in which they differ.

Uniform Mutation In uniform mutation, we select a parent, p, and replace a randomly selected gene $c_i \in p$ with a uniformly random number $c_i' \in [u_i, v_i]$ where u_i and v_i are set bounds.

Crossover Crossover is a binary operation, taking two randomly selected *parents* from the population P_t with a probability $p_c \in [0, 1]$

There are two major forms of crossover for binary coded GAs: $n > 1$ point crossover and Uniform crossover.

For real-coded GAs you instead have a selection of Crossover operators including:

- Flat crossover
- Simple crossover
- Whole arithmetic crossover
- Local arithmetic crossover
- Single arithmetic crossover
- BLX-α crossover

Simple (one point) Crossover For simple crossover, randomly select a crossover point, $i \in \{1, \ldots, n\}$. All values before this point are swapped between the two parents. For 2 parents, $p_1 = \left\{x_1^{[1]}, \ldots, x_n^{[1]}\right\}$ and $p_2 = \left\{x_1^{[2]}, \ldots, x_n^{[2]}\right\}$ this can be represented as:

$$p_1' = \left\{x_1^{[1]}, x_2^{[1]}, \ldots, x_i^{[1]}, x_{i+1}^{[2]}, \ldots, x_n^{[2]}\right\} \tag{2.2}$$

$$p_2' = \left\{x_1^{[2]}, x_2^{[2]}, \ldots, x_i^{[2]}, x_{i+1}^{[1]}, \ldots, x_n^{[1]}\right\} \tag{2.3}$$

Evaluation

After developing potential new individuals, the fitness of these new individuals is calculated using the objective function, $\mathcal{F}$

Reproduction

Here the next generation of individuals is constructed. There are multiple potential methods employed here with the simplest being to just replace the lest fit individuals with additional copies of the fitter individuals, be they pre-existing or newly generated.

In this stage various heuristics or alternative strategies can be implemented to speed up or slow down the convergence rate. Whilst their fitness

may be low, having a diverse population allows for more of the search space to be explored. If the algorithm converges too quickly it may get *stuck* in local minima.

2.1.4 Bézier Curves

In my implementation, I utilise Bézier curves to encode complex route arcs between a series of points. Here I will briefly outline their construction and mathematical basis.

Bézier curves were popularised by and named after French auto-body[3] designer Pierre Bézier in the 1960s and are commonly found in computer graphics today. They are parametric curves made up of a series of *control points* which contort the shape of a line to produce a curve of nearly any shape. Although their mathematical basis was established in 1912 by Sergei Bernstein in the form of Bernstein polynomials[3].

A Bézier curve is said to have a degree of $n-1$ where n is the number of control points including the start and end point of the curve.

Formal Definition

Bézier curves can be simply explicitly defined for degrees of 1 through 4, however, the general case of n points is more useful to us. In their general form they can be defined recursively or explicitly. I implement the recursive version as it is much more readable for a programming language that supports recursion.

They are defined recursively as follows:

$$\mathbf{B}_{P_0}(t) = P_0 \tag{2.4}$$

$$\mathbf{B}_{P_0,P_1,\ldots,P_n}(t) = (1-t)\mathbf{B}_{P_0,P_1,\ldots,P_{n-1}}(t) + t\mathbf{B}_{P_1,P_2,\ldots,P_n}(t) \tag{2.5}$$

Where the parameter t is in the range of $[0, 1]$. Therefore, the smoothness of the curve is determined by the granularity of the parameter when realising the curve.

2.2 Fully Autonomous Road Networks

Fully autonomous road networks are by no means a novel concept. They have appeared in fiction since the mid 20th century and have been a real possibility for the past decade. In a fully autonomous road networks *(FARN)*s, all vehicles are self-operating with their routing either being determined on a vehicle-by-vehicle *selfish* basis or by a larger system managing routes for all nearby agents.

[3] so I feel it is quite fitting that they find use in 21^{st} century automotive problems

In the latter system protocols that promote the net increase in efficiency can be implemented. However, this sort of system will require a form of government mandate as a universal routing protocol would need to be established and implemented by all auto manufacturers; this is no small undertaking.

2.3 Quantum Genetic Algorithms

2.4 Alternative Technologies

Chapter 3

Literature Review

3.1 Classical GAs

3.2 Fully Autonomous Road Networks

*FARN*s have the possibility of improving travel in a number of different areas.

They have the potential of greatly reducing travel times by efficiently routing all vehicles with the goal of reducing the net travel time. Such a system would also be able to respond much faster than human drivers, allowing for large increases in permissible vehicle velocities thus, further increasing efficiency.

By extension of their higher efficiency, *FARN*s will also have a marked effect on vehicular energy consumption. The potential effects are summarised nicely by Vahidi and Sciarretta[4] where they show the use of vehicle automation could cause anywhere from a halving of "energy use and greenhouse gas emission" in an "optimistic scenario" to doubling them depending on the effects that present themselves. The increase in highway speed whilst increasing travel efficiency is predicted by Brown et al.[5] and Wadud et al.[6] as increasing energy use by anywhere from 5% to 30%

3.3 Quantum GAs

3.3.1 Quantum Computing

Chapter 4

Classical Approach

4.1 Approach

The general shape of a GA as seen in Algorithm 1 is the same for almost all problems.

As such, I first had to implement a method to generate an initial population. In order to do this, I first had to define my programmatic representations of the Phenotypes and genotypes of my individuals. I decided to base my genotypic structure on the that described by Kala[2].

In this approach, the genotype is abstractly represented as a Bézier curve. This is more concretely translated into a real-valued string, interleaving the x and y coordinates of each control point.

Initially my Phenotype contained no additional information. However, I still created the distinction to allow for additional properties to be encapsulated during development.

4.2 Implementation

My for my initial implementation, I focused on simply getting a working system. I therefore, started by implementing the simplest form of each genetic operator. Namely, roulette wheel (Monte Carlo[7]) selection, simple crossover and uniform mutation.

4.2.1 Language Choice

I have chosen to implement my classical approach using the Julia language project[8].

Julia is a relatively new language first developed in 2012 by Jeff Bezanson, Stefan Karpinski and Viral Shah. It is a multi-paradigm language allowing for functional, object oriented and meta programming approaches to problems. I will mainly be using it for it's functional and OO capabilities.

Julia operates using multiple dispatch similar to languages such as Haskell. It interoperates with C and Fortran codebases without the need of middle-man bloat. This fact allows it to utilise the extensive high performance C libraries for floating point operations. Julia is eagerly evaluated, uses a Just in time compiler and has a garbage collector.

Julia features a syntax similar to both Python and Matlab with performance on par with C. As I am already very familiar with python and have studied functional programming in a number of modules; I found this language very quick and intuitive to learn and the resulting code to be clean, idiomatic and fast.

A real-world implementation of a system based on my research would undoubtedly be required to run on small, relatively low performance, embedded systems and as such Julia may not be appropriate here. A language such as C or Rust may be used instead.

4.3 Results

Chapter 5

Quantum Approach

5.1 Approach

5.2 Implementation

5.3 Results

Chapter 6

Evaluation

Chapter 7

Conclusion

Bibliography

[1] John H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. MIT Press, Cambridge, UNITED STATES, 1992.

[2] Rahul Kala. *On-Road Intelligent Vehicles: Motion Planning for Intelligent Transportation Systems / Rahul Kala.* Butterworth-Heinemann is an imprint of Elsevier, Kidlington, Oxford, UK, 2016.

[3] SN Bernstein. On the best approximation of continuous functions by polynomials of a given degree. *Comm. Soc. Math. Kharkow, Ser*, 2(13):49–194, 1912.

[4] Ardalan Vahidi and Antonio Sciarretta. Energy saving potentials of connected and automated vehicles. *Transportation Research Part C: Emerging Technologies*, 95:822–843, October 2018.

[5] Austin Brown, Jeffrey Gonder, and Brittany Repac. An Analysis of Possible Energy Impacts of Automated Vehicles. In Gereon Meyer and Sven Beiker, editors, *Road Vehicle Automation*, Lecture Notes in Mobility, pages 137–153. Springer International Publishing, Cham, 2014.

[6] Zia Wadud, Don MacKenzie, and Paul Leiby. Help or hindrance? The travel, energy and carbon impacts of highly automated vehicles. *Transportation Research Part A: Policy and Practice*, 86:1–18, April 2016.

[7] N Metropolis. The beginning. *Los Alamos Science*, 15:125–130, 1987.

[8] The Julia Programming Language. https://julialang.org/.